Ref. No.: 23402.8572

SET-ORIENTED REAL-TIME DATA PROCESSING BASED ON TRANSACTION BOUNDARIES

Inventor:
Gautam H. Mudunuri
Raymond G. To

BACKGROUND OF THE INVENTION

FIELD OF THE INVENTION

[0001] The present invention relates in general to data processing and database management. Specifically, the present disclosure relates to real time data processing in a database management system (DBMS) and particularly in Extract, Transformation, and Load (ETL), enterprise application integration (EAI), and enterprise information integration (EII). More specifically, the present disclosure provides set-oriented data transformation based on transaction boundaries.

DESCRIPTION OF THE RELATED ART

[0002] Many types of data processing operations associated with a DBMS or through ETL are batch-oriented. The batch-mode processing is performed after all data records in a data source are read and the output is produced only after the entire batch of data records is processed. The batch-mode transformation introduces intrinsic delays in data processing. Such delays become more

problematic in cascades of multiple transformations that involve multitudes of data records or rows, as the delay at one step would undermine the timeliness or responsiveness of the overall processing.

[0003] One example of batch-mode transformation can be found with aggregation, which is typically performed across all input data in a batch. In a wholesale setting, for instance, a number of ORDERS arrive in a message queue. Each ORDER is included in one message and the ORDER contains multiple LINEITEMs. To compute the sum of the dollar amount of all LINEITEMs in each ORDER, a batch-oriented system would calculate the sum across all ORDERS in the system. If an ORDER_ID uniquely identifying each ORDER is available, the sum within each ORDER could be computed by using the ORDER_ID as the group by key for the aggregation. Even if this were the case, the aggregation results may not be available until all the input data has been read. This is because batch-oriented systems typically presume that aggregation is to be performed across all input data. Similar situations will result in other multi-row transformations such as sorter, joiner, ranking, etc. The batch mode processing thus makes it nearly impossible to achieve the real-time responsiveness.

[0004] A need for data processing on a real-time basis is nonetheless conspicuous. For example, a retail store needs to know the up-to-minute summary of inventories; a postal mail service needs to provide customers with up-to-minute reports on their shipments; a financial service institution needs to continuously update its investors on the changes to their portfolios; and, a healthcare institution

needs to provide access to physicians and insurance administrators the updated patient information and most current prognosis reports.

[0005] There is a further need to break the batch-mode processing and substitute therefor, the data processing based on smaller data sets. The set-oriented processing would eliminate the delay of waiting on the entire batch to be read and processed. In the above example, the aggregation results for each data set – each ORDER – may be made available immediately in a set oriented real-time data processing system.

[0006] There is a still further need for a methodology to define transaction boundaries among the input data and thereby deriving data sets based on the applicable rules or business logic for data transformation on a real-time basis.

SUMMARY OF THE VARIOUS EMBODIMENTS

[0007] The present invention overcomes the limitations of batch-mode data processing by setting and propagating transaction boundaries for data transformation and thereby enables set-oriented data processing in real time. The invention supports cascades of data transformations on multitudes of data records and rows based on synchronized transaction boundaries. The invention may be implemented in conjunction with a relational DBMS, object DBMS, or any other DBMS. It may be advantageously applied in ETL, EAL, and EII.

[0008] In one embodiment, a method for processing a plurality of data records is provided. The method comprises: setting transaction boundaries among

the plurality of data records thereby dividing the plurality of data records into one or more data sets; processing each of the data set thereby producing a multiplicity of results from the one or more data sets; and completing the processing of the plurality by synchronizing the transaction boundaries and combining the multiplicity of results.

[0009] In another embodiment, a method for performing a series of transformations on a plurality of data records is provided. The series of transformations initiate at a source and conclude at a target. The method includes the steps of: setting transaction boundaries among the plurality of data records at the source thereby dividing the plurality of data records into one or more data sets; propagating the transaction boundaries through the series of transformations from the source to the target; performing the series of transformations based on the one or more data sets thereby producing a multiplicity of results from the series of set-based transformations; and completing the series of transformations by synchronizing the transaction boundaries and combining the multiplicity of results.

[0010] In accordance with this disclosure, there is provided, in yet another embodiment, a system for processing a plurality of data records. The system has: means for setting transaction boundaries among the plurality thereby dividing the plurality into one or more data sets; means for processing each of the data set thereby producing a multiplicity of results from the one or more data sets; and means for synchronizing the transaction boundaries and combining the multiplicity of results thereby completing the processing.

[0011] In accordance with this disclosure, there is provided, in still another embodiment, a system for performing a series of transformations on a plurality of data records. The series of transformations initiate at a source and conclude at a target. The system has: means for setting transaction boundaries among the plurality at the source thereby dividing the plurality into one or more data sets; means for propagating the transaction boundaries through the series of transformations from the source to the target; means for performing the series of transformations based on the one or more data sets thereby producing a multiplicity of results from the series of set-based transformations; and means for synchronizing the transaction boundaries and combining the multiplicity of results thereby completing the series of transformations.

[0012] In one embodiment, the transaction boundaries are defined based on the row count of the data records. In another embodiment, the transaction boundaries are defined based on the time stamp of the data records. In yet another embodiment, the transaction boundaries are defined based on the result of a previous data transformation.

[0013] In still another embodiment, the transaction boundaries are defined based on a user-defined logic. The user-defined logic is one or more rules defined by a user. In a further embodiment, the user-defined logic is on a real-time basis. In a still further embodiment, the rules comprise one or more tables in a database. In another embodiment, the rules comprise one or more statements defining relationships and actions in a suitable programming language. In yet another

embodiment, the suitable programming language is one of Generation III Languages (3GL), Generation IV Languages (4GL), and Generation V Languages (5GL). In still another embodiment, the suitable programming language is an expert system tool.

[0014] In a further embodiment, the means for propagating the transaction boundaries maintains one or more transaction queues capable of defining the boundaries of the data sets. In a still further embodiment, the transaction queues comprise one or more tables in a database. In another embodiment, the transaction queues are maintained in a computer memory.

[0015] In still another embodiment, the series of transformations comprise at least one of insert, update, delete, aggregation, rank, sort, sequence, and join.

BRIEF DESCRIPTION OF THE DRAWINGS

[0016] Fig. 1 is a flow chart showing the steps of set-based data transformations according to one embodiment.

[0017] Fig. 2 is a block diagram of a system for set-based data transformation in one embodiment.

[0018] Fig. 3 shows an overview of the mappings of relevant tables and data transformations for answering a database query according to one embodiment.

[0019] Fig. 4 shows the transformation of sorting the data records according to one embodiment.

[0020] Fig. 5 shows how transaction boundaries are defined and set according to one embodiment.

[0021] Fig. 6 shows various data ports or variables of a condition-checking transformation that defines transaction boundaries according to one embodiment.

[0022] Fig. 7 shows the attributes of a transaction control transformation that sets transaction boundaries according to one embodiment.

[0023] Fig. 8 shows various data ports of an aggregator transformation according to one embodiment.

[0024] Fig. 9 is shows the attributes of the aggregator transformation of Fig. 8 according to one embodiment.

[0025] Fig. 10 shows the attributes of a joiner transformation that joins transactions based on preserved transaction boundaries according to one embodiment.

[0026] Fig. 11 shows a filter transformation, which allows the transaction boundaries in the data records to pass through and propagate onto the target according to one embodiment.

[0027] Fig. 12 shows the attributes of the filter transformation of Fig. 11 according to one embodiment.

[0028] Fig. 13 shows an overview of the mappings of relevant tables and the series of transformations from a XML document source according to one embodiment.

[0029] Fig. 14 shows the cascades of transformations for reading and parsing XML document input according to one embodiment.

[0030] Fig. 15 shows how transaction queues propagate and preserve the transaction boundaries through cascades of transformations according to one embodiment.

[0031] Fig. 16 shows the attributes of the XML parsing transformation according to one embodiment.

[0032] Fig. 17 shows the attributes of the summary aggregator transformation according to one embodiment.

[0033] Fig. 18 shows how the results from the join transformation are loaded onto the target according to one embodiment.

[0034] Fig. 19 shows the attributes of a summary join transformation according to one embodiment.

DETAILED DESCRIPTION OF THE VARIOUS EMBODIMENTS

Discussion of the Relevant Terms

[0035] The following terms, data, data stream, relational database, database management, programming languages, table, record, row, column, join, sort, rank, aggregation, commit, roll back, as well as other relevant terms throughout the present disclosure, are to be understood consistently with their typical meanings established in the relevant art, *i.e.* the art of information technology, computer sciences, mathematics, statistics, data processing, and knowledge management.

[0036] Transformation, as used herein, refers to a form of data processing that derives one or more sets of output rows or records from one or more sets of input rows or records. In certain embodiments, "transformation" is used interchangeably with "data processing." For example, in the context of a DBMS, whether a relational DBMS or an object DBMS, "transformation," "data transformation," and "data processing" are used synonymously.

[0037] Source or data source, as used in this disclosure, refers to the data input as well as the system where the input data is stored. A source may include one or more input table or one or more rows and records from the input table. A source may be a relational database, an object database, a data warehouse, a file, a message queue, or other data sources. In certain embodiments, "source" and "source system" are used interchangeably.

[0038] Target or data target, as used in this disclosure, refers to the data output as well as the system where output data is loaded. A target may include one or more output table or one or more rows and records from the output table. A target may be a relational database, an object database, a data warehouse, a file, a message queue, or other target data depositaries. In certain embodiments, "target" and "target system" are used interchangeably.

[0039] Mapping, as used herein, refers to the correspondence between the input data or the source and output data or the target, as well as the transformation from the source to the target. A mapping may include multiple sources and targets.

[0040] Generation III Languages (3GL), as used herein, refer to high-level programming languages, such as PL/I, C, C++, or Java. Typically, a compiler converts the statements of a specific high-level programming language into machine language. According to certain embodiment, 3GLs encompass script languages such as Perl, JavaScript, Visual Basic, and Tcl/Tk.

[0041] Generation IV Languages (4GL), as used herein, refer to the programming languages that are designed to be closer to natural language than a 3GL language. Languages for accessing databases, such as SQL statements, belong to the 4GLs.

[0042] Generation V Languages (5GL), as used herein, refer to programming languages that use a visual or graphical development interface to create source language that is usually compiled with a 3GL or 4GL language compiler. For

example, companies like Microsoft, Borland, IBM, and others make 5GL visual programming products for developing applications in Java, and C. With 5GLs, a developer may visualize class hierarchies and drag icons to assemble program components.

[0043] Data set, as used in this disclosure, refers to one or more contiguous records or rows grouped together for processing. Data sets define logical transformation boundaries in a data stream. Data sets may be set according to row counts (e.g., a set may be defined as *n* rows or records in a source table), time (e.g., a set may be defined as all the rows or records within a given period of time), or other rules including user-defined logic. According to one embodiment, data sets do not overlap with each other. A data set may contain multiple nested datasets, which are non-overlapping at each level.

[0044] Transaction set, as used in this disclosure, refers to one or more data records grouped together for processing. A transaction set may be used interchangeably with a data set in certain embodiments. A transaction set may contain one or more data sets in alternative embodiments. When data transactions are performed based on a data set, the data set becomes a transaction set. A transaction set boundary must coincide with a data set boundary.

[0045] A transaction occurs when the changes to the data records are committed or rolled back. In certain embodiments, the data processing operates on a set without an explicit commit or rollback after the end of the data stream. This is still considered as a complete transformation or transaction of the data set.

A transaction set is transactionally applied to a target system. That is, the changes are considered to be simultaneous among the records or rows in the transaction. Transaction sets define logical transaction boundaries in a data stream. Transaction sets may be set according to row counts, time, or other rules including user-defined logic. Transaction sets are non-overlapping in one embodiment. Detailed discussions of data sets and transaction sets are set forth below.

Transaction Boundaries Defined to Support Real Time Data Processing

[0046] The present disclosure provides a mechanism for defining transaction boundaries and thereby grouping multiple rows of data records into data sets or transaction sets. Transformations based on smaller data sets – as opposed to the entire batch of data – afford improved flexibility and timeliness in the overall data processing. The output of results on the smaller data sets is made immediately available upon completion of each transaction defined by the transaction boundaries. Such immediacy substantiates the real-time responsiveness to data queries. The delay in batch-oriented transformation caused by reading and processing all the rows in the entire batch of data source is eliminated or substantially reduced.

[0047] In one embodiment, a real-time or near real-time multi-row data transformation is performed by setting transaction boundaries among a plurality of data records, processing the data sets defined by the transaction boundaries, and combining the results from the set-based processing. The plurality of records may

encompass multiple rows of data, including rows of interrelated data records. For example, a plurality of records can encompass one ORDER record and multiple LINEITEM records that belong to the ORDER. In other words, the plurality of records includes multiple records of one or more record types.

[0048] Referring to Fig. 1, the input data 101 contains multiple rows of data records. Transaction boundaries are defined and set (103) within the data records (101) to produce a collection of data sets or transaction sets (105). Data processing or transformation (107) is then performed based on the data sets (105). The results from set-based transformation are then combined (109) to produce the final results (111).

[0049] In another embodiment, a series of transformations is performed on multi-row data records by setting transaction boundaries among multi-row data records, propagating the transaction boundaries through the series of transformations, processing the data sets defined by the transaction boundaries, and combining the results from the set-based processing. The series of transformations may be performed on real time basis based on certain rules or user-defined logic.

[0050] In yet another embodiment, a system is provided for real-time or near real-time data processing of multi-rows of data records. The system includes means for setting transaction boundaries in multi-row data records, means for processing data sets based on transaction boundaries; and means for synchronizing the transaction boundaries and combining the results of set-based transformations.

[0051] For example, referring to Fig. 2, a database 201 is connected to a data processing system or unit 203. Also linked to the data processing system (203) is a module (205) that is adapted to setting transaction boundaries, a module (207) that is adapted to propagating transaction boundaries, and another module (209) that is adapted to synchronizing the transaction boundaries and combining the results from set-based transformations. The modules 203, 205, and 207 thus provide an extension of the data processing system (203) and enable the set-based data transformation on the data records stored in database (201).

[0052] A system is provided for performing a series of transformations on a plurality of data records. The system includes means for setting transaction boundaries among multi-row data records, means for propagating the transaction boundaries through the series of transformations, means for performing the series of transformations based on data sets, and means for synchronizing the transaction boundaries and combining the results of set-based transformations.

[0053] The data processing systems may be implemented as software or hardware solutions in various embodiments. The enumerated means constituting the data processing systems in the various embodiments may be a software module, a firmware element, or a hardware component. Computer program products that implement the set-oriented real-time data transformation systems are provided in one embodiment. Also provided are computer readable media having recorded thereon program instructions which when processed by a computer are

capable of executing the method of set-oriented real-time data transformation according to this disclosure.

[0054] Transaction boundaries are set to break the input data stream into data sets for transformation. Such transaction data sets are defined based on applicable rules or logic in a given situation. Simply applying real-time flushes cannot be used to set logical transaction boundaries, as they merely impose a latency control and guarantees no consistency.

[0055] According to the present disclosure, the means for setting transaction boundaries defines the transaction boundaries based on (i) the row count of the data records in one embodiment; (ii) the time stamp of the data records in another embodiment (e.g., the time when the data records become available); and, (iii) the result of a previous data transformation in yet another embodiment. The multiple and series of transformations may thereby synchronize and coordinate the transaction boundaries through the set-based transformations and combine the results thereof to complete the entire data processing (see below, example 2).

[0056] In a further embodiment, the means for setting transaction boundaries comprises defining the transaction boundaries based on a user-defined logic. The user-defined logic is one or more rules defined by a user. The rules may be implemented in one or more tables in a database or one or more statements defining relationships and actions in a suitable programming language. The suitable programming language includes a 3GL, 4GL, and 5GL. The suitable programming language may be an artificial intelligent programming language

such as an expert system tool. The examples include Common LISP and CLIPS (C Language Integrated Production System).

[0057] In a further embodiment, the user-defined logic defines rules for real-time processing. For example, the user-defined rules may specify a short time interval for processing data sets that arrive at a source of message queue thereby achieving the real-time responsiveness. See below, example 2. The user-defined rules may specify the transaction boundaries wherever a user report is scheduled thereby masking the latency of processing time and producing a real-time effect.

Multi-row Transformations Based on Data Sets

[0100] Data transformations are performed based on data sets or transaction sets according to the methods and systems of this disclosure. Data is processed within transaction boundaries. Where the real-time responsiveness is desired and supported, the transactions in one embodiment flush the data through the mappings or series of transformations to the target, rather than buffering it up for performance reasons. This assures that the data is transformed and loaded in real-time.

[0101] Methods or system means for general data processing and transformation in a database, whether a relational, object, or another DBMS, are known to an ordinarily skilled database engineer, application developer, or a database administrator. Examples of data transformations include insert, update, delete, aggregation, rank, sort, join, XML parser, and XML generator. The set-

oriented data processing according to this disclosure adopts the data processing means for insert, update, delete, aggregation, rank, sort, join, XML parser, XML generator, among others, and applies the same to data set-based transformation.

[0102] For example, aggregation typically operates on the entire input data sets. An unsorted aggregator computes and outputs aggregates for the current transformation dataset, i.e., the entire outstanding input set. An unsorted aggregator resets its aggregation cache so that data from the current transformation dataset does not affect the aggregation results for any subsequent transformation dataset. The variable ports are also reset to their initial values. A sorted aggregator computes and outputs aggregates for the current transformation input data by key. It resets its state for a new input set by key for the first group in the next transformation dataset. The variable ports are also reset to their initial values.

[0103] In one embodiment, set-based aggregation or incremental aggregation is provided. It applies to unsorted aggregation, for example. The aggregation may be incremental across session runs or transactions. Each set transformation may in effect constitute a separate session run or transaction. The aggregates are output at the end of a transaction boundary. The variable ports are reset to their initial values based on the transaction boundaries.

[0104] Rank may be performed in a set-oriented mode according to one embodiment. It outputs the rank results for each transaction dataset. It re-initializes its cache in preparation for ranking the next dataset. The variable ports

are also reset to their initial values. In other embodiments, rank operates in one mode to process the current input transformation set.

[0105] Set-oriented sorter may also be supported according to another embodiment. When the sorter receives transaction boundaries, it sorts and outputs rows for each of the data sets or transaction sets. It clears out its buffers (in memory or on disk) in preparation for sorting the next dataset. In alternative embodiments, the sorter does not implement the "incremental sort" functionality. The sorted output for each transformation input set includes data from all previous transformation input sets, and thus no need for sorting smaller transaction sets separately.

[0106] According to a further embodiment, joiner is performed based on data sets or transaction sets. Various types of joins are supported, including sequential join, concurrent join, nested-loop join, and sorted-merge join, among others.

[0107] Self join, or same-virtual source join, is a join where the master and detail inputs of the joiner come from a single source or a single virtual source. For example, a multi-group data source or a transformation with multiple output groups may be self-joined based on the data sets or groups. See below, example 2. Such data sets or groups may be treated as transaction sets. They may result from a previous transformation that is based on transaction sets. In one embodiment, the same virtual source join is defined for the concurrent sort-merge joiner. Since both the inputs come from the single source of transformation datasets, the

resulting datasets of the join correspond directly to the datasets of the virtual source. The outer-joins are done within each input dataset.

[0108] Whether the join is from the same group of a source, or from multiple groups of the same source, transaction boundaries are set at the source. These boundaries propagate in the data stream on both inputs to the concurrent sort-merge joiner, for example. When the concurrent sort-merge joiner receives the corresponding set boundaries on both inputs, it first performs outer join (if specified) as if an End-of-File (EOF) was received on both inputs (thus processing the set as a typical data batch), and next resets its state for outer joins in preparation for the next data set.

[0109] A different-source join receives data inputs from different sources. It enables set-oriented processing in certain embodiments and in other embodiments only supports batch-mode transformation. For the sequential sorted-merge join, sequential nested-loop join, and concurrent nested-loop join, two source inputs are typically provided: a master and a detail. The master cache needs to be populated fully before the detail can be processed. The detail input may support real-time transformation based on transaction sets. The only state for the detail input is the outer join (if specified) results. The outer join rows can be output at the end of each of the detail datasets.

[0110] In a still further embodiment, the sequence transformation may be performed based on transaction sets. It has a "reset" flag. When the flag is turned on, the counter value for the sequence is reset to its initial value at each of the

transaction boundaries in the input data. Because the same sequence may be connected to multiple transformations, if these transformations are running concurrently, the sequence numbers they receive will be from one shared counter. Resetting the sequence by the consumer may interfere with the sequence values delivered to other transformations. Thus, in another embodiment, the sequence only resets at the transaction boundaries when it is not feeding concurrent transformations. In alternative embodiments, the sequence transformation does not respond to transaction sets; it operates based on the entire input data set.

[0111] An expression transformation supports set-oriented processing in certain embodiments. It resets all the variable ports to their initial values when receiving a transaction boundary.

[0112] In still another embodiment, transformations in a "pull pipeline" may be performed based on transaction sets. A "pull pipeline" originates from a sequence transformation and concatenates into a transformation along with a pipeline that originates from a source transformation. The pull pipeline transformation responds to transaction data set boundaries arriving at the point of the concatenation of the pipeline. For example, if the pull pipeline has an expression transformation, its variable ports may be reset.

Preservation of Transaction Semantics through Cascades of Transformations

[0113] The method and system of this disclosure enables real-time data transformation based on data sets or transaction sets. The set-oriented mode is

supported in cascades of multiple transformations. Each transaction boundary in the input data stream is propagated through the cascades of transformations and thereby preserving the transaction semantics from the data source to the target.

[0114] The results from each transaction set are combined with the results from the other transaction sets, or fed into the input of a subsequent transformation to complete the series of transformations. Transaction sets are synchronized or coordinated when the results of multiple set-based transformations are combined.

[0115] In one embodiment, the propagation of the transaction boundaries is based on one or more transaction queues set and maintained in a computer memory or a database. A transaction queue is capable of defining the boundaries of the data sets or transaction sets in the input data stream. Each transaction for the series set-based transformations may be associated with and informed by a specific transaction queue. The transaction queue in one embodiment is maintained in a computer memory of a data processing system. In another embodiment, the transaction queue includes one or more tables in a database. The transaction queues set forth the start and end rows of each data set or transaction set. See, e.g., example 2 below.

[0116] The multi-row transformations based on preserved transaction boundaries according to certain embodiments may be implemented in conjunction with a database management system, such as a relational DBMS, object DBMS, or any other DBM, a text file, a XML document, a message queue, and an Enterprise Resource Planning (ERP) system. The method and system for such set-oriented

transformations according to one embodiment may be applied in ETL, EAI, and EII to enhance the flexibility and real-time responsiveness of data processing and dissemination in an enterprise setting thereby facilitating effective information management.

[0117] The various embodiments are further described by the following examples, which are illustrative of the disclosed embodiments but do not limit the same in any manner.

Example 1: Transaction Boundaries in Employee Records Data

[0118] This example describes the data processing steps for answering a query to a database of employee records. Transaction boundaries are defined according to one embodiment. Set-based transformations are performed to produce the final results.

[0119] Consider the query: given a list of employee records, find all employees that earn more than the average salary in their respective departments.

[0120] An overview of the mappings of relevant tables in the database and a number of data transformations for this query is shown in Fig. 3. Certain reference numerals in Fig. 3 correspond to the graphical components or transformations in the subsequent Figs 4-10.

[0121] Referring to Fig. 3, the employee (*emp*) source file (101) contains the following fields:

Employee ID (*empid*).
Department ID (*deptid*).

Employee Name (*ename*).
Salary (*sal*).

[0122] Suppose the following are the input data from the *emp* source file (101):

```
empid, deptid, ename, sal

1,2,'John',80000
2,1,'Jack',75000
3,3,'Sally',85000
4,1,'William',65000
5,3,'Sue',45000
5,2,'Mark',82000
7,2,'Bart',74000
8,1,'Steven',96000
9,3,'George',56000
9,1,'Brenda',86000
10,3,'Sam',49000
11,2,'Anne',79000
11,1,'Jill',68000
12,3,'Patrick',77000
```

[0123] The first step is to read (201) the source file (101) and sort it on *depid* as the sort key (203), as shown in Fig. 4. The result of the sorting is as follows:

```
empid, deptid, ename, sal

2,1,'Jack',75000
4,1,'William',65000
8,1,'Steven',96000
9,1,'Brenda',86000
11,1,'Jill',68000
1,2,'John',80000
5,2,'Mark',82000
7,2,'Bart',74000
11,2,'Anne',79000
3,3,'Sally',85000
5,3,'Sue',45000
9,3,'George',56000
10,3,'Sam',49000
12,3,'Patrick',77000
```

[0124] Next, the employee records are divided into multiple transaction or data sets according to the different departments the employees belong to. That is, referring to Fig. 5, transaction boundaries are set (303) where the *deptid* changes (301). The *deptid_changed* expression transformation (301) determines if the *deptid* has changed by comparing the current value of *deptid* with the value from the previous row. The result of the change is output on a Boolean port *deptid_changed*. Fig. 6 shows the various data ports or variables of the *deptid_changed* expression transformation (301) according to one embodiment. The *tx_deptid* transaction control transformation (TCT) (303) uses the *deptid_changed* to insert a *COMMIT* before the row with a changed *deptid*. Fig. 7 shows the attributes of the *tx_deptid* transaction control transformation (303) according to one embodiment.

[0125] The employee data is thus divided into three transaction or data sets. The resulting data stream with the embedded commits is as follows:

```
empid, deptid, ename, sal

2,1,'Jack',75000
4,1,'William',65000
8,1,'Steven',96000
9,1,'Brenda',86000
11,1,'Jill',68000
<COMMIT>
1,2,'John',80000
5,2,'Mark',82000
7,2,'Bart',74000
11,2,'Anne',79000
<COMMIT>
3,3,'Sally',85000
5,3,'Sue',45000
9,3,'George',56000
```

```
10,3,'Sam',49000
12,3,'Patrick',77000
```

[0126] The next step is to compute the average of salary using the *avg_sal* aggregator transformation (103). The input to the aggregator is the *deptid* and *sal* columns only. Fig. 8 shows the various data ports or variables of the *avg_sal* aggregator transformation (103). Fig. 9 shows the attributes of the *avg_sal* aggregator transformation (103). The transaction boundaries are propagated into this transformation, shown as follows:

```
deptid, sal

1,75000
1,65000
1,96000
1,86000
1,68000
<COMMIT>
2,80000
2,82000
2,74000
2,79000
<COMMIT>
3,85000
3,45000
3,56000
3,49000
3,77000
```

[0127] According to one embodiment, the transformation scope of the aggregator is set to allow set-based transactions. The aggregator thus transforms within the input transaction boundaries. The results of aggregation are as follows, with the transaction boundaries preserved at the output.

```
deptid, avg_sal

1,78000
```

```
<COMMIT>
2,78750
<COMMIT>
3,62400
```

[0128] Further, the aggregator results are joined to the original data using the *join_deptid* joiner transformation (901), referring to Fig. 11. The join is performed on *deptid*. Transaction boundaries are further propagated to the joiner transformation. Fig. 10 shows the details of an editor (801) for defining and editing the *join_deptid* joiner transformation (901) according to one embodiment. The joiner synchronizes the transactions coming into its two input streams based on the transaction boundaries, joins within the transactions, and outputs the resulting data with transaction boundaries preserved.

[0129] The results of join are as follows:

```
empid, avg_sal, deptid, ename, sal

2,78000,1,'Jack',75000
4,78000,1,'William',65000
8,78000,1,'Steven',96000
9,78000,1,'Brenda',86000
11,78000,1,'Jill',68000
<COMMIT>
1,78750,2,'John',80000
5,78750,2,'Mark',82000
7,78750,2,'Bart',74000
11,78750,2,'Anne',79000
<COMMIT>
3,62400,3,'Sally',85000
5,62400,3,'Sue',45000
9,62400,3,'George',56000
10,62400,3,'Sam',49000
12,62400,3,'Patrick',77000
```

[0130] Finally, the data records are processed to filter out (903) those with the *sal* value lower than *avg_sal*. Fig. 12 shows the details of an editor (1001) for

defining and editing the filter transformation (903) according to one embodiment. The results from the filter transformation (903) are then loaded to the target (905). Because the filter (903) is a transformation with a row scope (i.e., it is not a set-oriented transformation), it simply allows the transaction boundaries in the data records to pass through and propagate onto the target.

[0131] The final results of this transformation are as follows:

```
empid, avg_sal, deptid, ename, sal

8,78000,1,'Steven',96000
9,78000,1,'Brenda',86000
<COMMIT>
1,78750,2,'John',80000
5,78750,2,'Mark',82000
11,78750,2,'Anne',79000
<COMMIT>
3,62400,3,'Sally',85000
12,62400,3,'Patrick',77000
```

[0132] The records are then loaded into the target:

```
empid, deptid, ename, sal

8,1,'Steven',96000
9,1,'Brenda',86000
<COMMIT>
1,2,'John',80000
5,2,'Mark',82000
11,2,'Anne',79000
<COMMIT>
3,3,'Sally',85000
12,3,'Patrick',77000
```

Example 2: Set-based Transformations from a Data Source of XML Document Records

[0133] This example describes a series of transformations from a data source of XML (Extensible Markup Language) document records. Transformations are performed within transaction boundaries, which are defined, propagated, and synchronized through cascades of transformations according to one embodiment.

[0134] Fig. 13 provides an overview of the mappings of the relevant tables and the series of transformations involved. Certain reference numerals in Fig. 13 correspond to the graphical components or transformations in the subsequent Figs 12-17.

[0135] Each source record consists of a single field representing an XML document, one document per row. Specifically, the source is a file (1101) with one XML document per line. Each XML document contains multiple ORDERS, where each ORDER contains multiple LINEITEMS. The object is to compute summary information, e.g., number of items and total amount, for each order and output the result to the target, the ORDER_SUMMARY table.

[0136] Suppose the source data contains the following two records:

[0137] First Record:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<XRoot xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <ORDERS>
    <LINEITEMS>
      <ITEM_ID>7</ITEM_ID>
      <ITEM_DESC>Lawn Mower</ITEM_DESC>
      <AMOUNT>499</AMOUNT>
    </LINEITEMS>
    <LINEITEMS>
      <ITEM_ID>9</ITEM_ID>
      <ITEM_DESC>Bathtub</ITEM_DESC>
      <AMOUNT>899</AMOUNT>
    </LINEITEMS>
```

```
    <LINEITEMS>
      <ITEM_ID>14</ITEM_ID>
      <ITEM_DESC>Entry Door</ITEM_DESC>
      <AMOUNT>549</AMOUNT>
    </LINEITEMS>
    <CUSTOMER_ID>1</CUSTOMER_ID>
    <CUSTOMER_NAME>John</CUSTOMER_NAME>
    <ORDER_ID>1</ORDER_ID>
    <ORDER_DATE>2003-11-04</ORDER_DATE>
  </ORDERS>

  <ORDERS>
    <LINEITEMS>
      <ITEM_ID>21</ITEM_ID>
      <ITEM_DESC>Patio Deck</ITEM_DESC>
      <AMOUNT>1049</AMOUNT>
    </LINEITEMS>
    <LINEITEMS>
      <ITEM_ID>17</ITEM_ID>
      <ITEM_DESC>Landscape Lighting</ITEM_DESC>
      <AMOUNT>699</AMOUNT>
    </LINEITEMS>
    <CUSTOMER_ID>2</CUSTOMER_ID>
    <CUSTOMER_NAME>Jack</CUSTOMER_NAME>
    <ORDER_ID>3</ORDER_ID>
    <ORDER_DATE>2003-10-23</ORDER_DATE>
  </ORDERS>

  <ORDERS>
    <LINEITEMS>
      <ITEM_ID>44</ITEM_ID>
      <ITEM_DESC>Steel Kitchen Sink</ITEM_DESC>
      <AMOUNT>1249</AMOUNT>
    </LINEITEMS>
    <LINEITEMS>
      <ITEM_ID>77</ITEM_ID>
      <ITEM_DESC>Platinum Shower Head</ITEM_DESC>
      <AMOUNT>199</AMOUNT>
    </LINEITEMS>
    <CUSTOMER_ID>9</CUSTOMER_ID>
    <CUSTOMER_NAME>Patrick</CUSTOMER_NAME>
    <ORDER_ID>4</ORDER_ID>
    <ORDER_DATE>2003-11-23</ORDER_DATE>
  </ORDERS>

</XRoot>
```

[0138] Second Record:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<XRoot xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <ORDERS>
    <LINEITEMS>
      <ITEM_ID>92</ITEM_ID>
      <ITEM_DESC>100 Pack Light Bulbs</ITEM_DESC>
```

```
      <AMOUNT>49</AMOUNT>
    </LINEITEMS>
    <LINEITEMS>
      <ITEM_ID>74</ITEM_ID>
      <ITEM_DESC>Giant Weed Killer</ITEM_DESC>
      <AMOUNT>19</AMOUNT>
    </LINEITEMS>
    <CUSTOMER_ID>2</CUSTOMER_ID>
    <CUSTOMER_NAME>Jack</CUSTOMER_NAME>
    <ORDER_ID>5</ORDER_ID>
    <ORDER_DATE>2003-11-24</ORDER_DATE>
  </ORDERS>

  <ORDERS>
    <LINEITEMS>
      <ITEM_ID>92</ITEM_ID>
      <ITEM_DESC>Water Purifier</ITEM_DESC>
      <AMOUNT>249</AMOUNT>
    </LINEITEMS>
    <LINEITEMS>
      <ITEM_ID>74</ITEM_ID>
      <ITEM_DESC>20 Gal White Paint</ITEM_DESC>
      <AMOUNT>399</AMOUNT>
    </LINEITEMS>
    <CUSTOMER_ID>11</CUSTOMER_ID>
    <CUSTOMER_NAME>Smith</CUSTOMER_NAME>
    <ORDER_ID>6</ORDER_ID>
    <ORDER_DATE>2003-11-27</ORDER_DATE>
  </ORDERS>

</XRoot>
```

[0139] First, the source data is read and the XML document is parsed using the *parse_orders* XML parser transformation (1201), referring to Fig. 14. The parsing is done one row at a time, because in this example the source is configured to issue transactions (source-based commits) every single row. Thus, each XML document constitutes a transaction set or data set in this embodiment.

[0140] In an alternative embodiment, if the data source is a time-sensitive or real-time data source such as a message queue that is being continuously updated, the parsing transformation may be performed base on time-driven rules or business logic. That is, for example, if each message contains a XML document,

and the outstanding messages are being processed at a given time interval (e.g., 10 seconds), the transaction boundaries can be set to be equal to or multiples of the time interval. Note that they are between messages and not within a message – this prevents us from processing a partial message, resulting in partial or incorrect results.

[0141] Referring to Fig. 14, *SQ_raw_orders* (1203) is a transaction generator for setting up transactions. A transaction queue (TQ, 1205) is created that associates with the generator (1203). Given that there are two input records in this example and that the transaction boundaries for commit are at every row, two transaction sets are derived. Hence, the transaction queue (1205) includes two entries. It lists the starting and ending rows of each transaction set.

[0142] The *parse_orders* XML parser transformation (1201) outputs a row for each ORDER and multiple rows of LINEITEM for each order. The LINEITEMs are associated with their corresponding ORDER via the ORDER_ID field.

[0143] Since the transformation scope of *parse_orders* XML parser transformation (1201) is row scope, as mentioned above, the input transactions and the transaction boundaries are propagated onto the output. Because each input row or each XML document constitutes a transaction set, the output of parsing propagates the same boundaries. That is, the transaction boundaries will enclose the results from parsing each XML document. The outputs of parsing are as follows for ORDERS and LINEITEM:

[0144] ORDERS

Row No	Customer_ID	Customer_Name	Order_ID	Order_Date
1	1	John	1	2003-11-04
2	2	Jack	2	2003-10-23
3	9	Patrick	3	2003-11-23
COMMIT				
4	2	Jack	4	2003-11-24
5	11	Smith	5	2003-11-27

[0145] LINEITEMS

Row No	Order_ID	Item_ID	Item_Desc	Amount
1	1	7	Lawn Mower	499
2	1	9	Bathtub	899
3	1	14	Entry Door	549
4	2	21	Patio Deck	1049
5	2	17	Landscape Lighting	699
6	3	44	Steel Kitchen Sink	1249
7	3	77	Platinum Shower Head	199
COMMIT				
8	4	92	100 Pack Light Bulbs	49
9	4	74	Giant Weed Killer	19
10	5	92	Water Purifier	249
11	5	74	20 Gal White Paint	399

[0146] Referring to Fig. 15, LINEITEMS is fed to the *calc_summary* aggregator transformation (1303), which computes the number of items and the total amount in an order. The transformation scope of *calc_summary* (1303) is set to "transaction" as opposed to "row," and thus the *calc_summary* transformation (1303) is performed based on transaction sets. It computes the summaries within the transaction boundaries. A transaction queue (1307) is created and maintained, which supports the *calc_summary* transformation (1303). The transaction queue

(1307) set forth the transaction boundaries, i.e., the start and end rows, of the *calc_summary* transformation (1303). The results from *calc_summary* are as follows

Row No	Order_ID	Num_Items	Total_Amount
1	1	3	1947
2	2	2	1748
3	3	2	1448
COMMIT			
4	4	2	68
5	5	2	648

[0147] The output of ORDER reaches the Master input of the *join_summary* joiner transformation (1211), referring to Fig. 15.

[0148] The summary results are then joined with the orders using the *join_summary* joiner transformation (1211). The *join_summary* joiner transformation uses ORDER_ID as the join key. The transformation scope of join_summary is set to "transaction" as opposed to "row," and thus the *join_summary* transformation is performed based on transaction sets. Two transaction queues (1305, 1309) are created and maintained to support the *join_summary* transformation (1301), providing the transaction boundaries, i.e., the start and end rows, for each of the two input streams of the joiner.

[0149] Finally, the following results from *join_summary* (1301) are loaded into the target (1601), as shown in Fig. 18. A transaction queue (1603) is also used to ensure that the transaction boundaries are preserved.

Row	Customer_	Customer_	Order_ID	Order_Date	Num_	Total

No	ID	Name			Items	Amount
1	1	John	1	11/04/2003	3	1947
2	2	Jack	2	10/23/2003	2	1748
3	9	Patrick	3	11/23/2003	2	1448
COMMIT						
4	2	Jack	4	11/24/2003	2	68
5	11	Smith	5	11/27/2003	2	648

[0058] The attributes, scope (set-based or batch-based), and data ports (variables) of the various transformations may be defined by a user through an interface as needed. For example, Fig. 16 shows the attributes of the *parse_orders* XML parser transformation (1201); Fig. 17 shows the attributes of the *calc_summary* transformation (1303); and, Fig. 19 shows the attributes of the *join_summary* transformation (1301).

[0150] It is to be understood that the description, specific examples and data, while indicating exemplary embodiments, are given by way of illustration and are not intended to limit the various embodiments of the present disclosure. All references cited herein are specifically incorporated by reference in their entireties. Various changes and modifications within the present disclosure will become apparent to the skilled artisan from the description, examples, and data contained herein, and thus are considered part of the various embodiments of this disclosure.